```
'''
Question 1: Using Pandas Perform Merge and Join.
'''
import pandas as pd

df1 = pd.DataFrame({
    'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})

df2 = pd.DataFrame({
    'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
    'salary': [70000, 80000, 120000, 90000]})

print("--- DataFrame 1 (df1) ---")
print(df1)

print("\n--- DataFrame 2 (df2) ---")
print(df2)

print("\n═══════════════════════════════════════════════════════════════════════")
print("Merge using left_on='employee' and right_on='name'")
merged_on_columns = pd.merge(df1, df2, left_on="employee",
right_on="name").drop('name', axis=1)
print(merged_on_columns)

# Merge using 'left_index' and 'right_index'
df3 = pd.DataFrame({
    'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
    'hire_date': [2004, 2008, 2012, 2014]
})

df1a = df1.set_index('employee')
df2a = df3.set_index('employee')

print("\n═══════════════════════════════════════════════════════════════════════")
print("--- (df1a) ---")
print(df1a)

print("\n--- (df2a) ---")
print(df2a)

print("Merge using left_index=True and right_index=True")
merged_on_index = pd.merge(df1a, df2a, left_index=True, right_index=True)
print(merged_on_index)


'''
Output:
--- DataFrame 1 (df1) ---
  employee        group
0      Bob   Accounting
1     Jake  Engineering
2     Lisa  Engineering
```

```
3      Sue           HR

--- DataFrame 2 (df2) ---
   name  salary
0   Bob   70000
1  Jake   80000
2  Lisa  120000
3   Sue   90000

================================================================================
Merge using left_on='employee' and right_on='name'
  employee        group  salary
0      Bob   Accounting   70000
1     Jake  Engineering   80000
2     Lisa  Engineering  120000
3      Sue           HR   90000

================================================================================
--- (df1a) ---
                group
employee
Bob         Accounting
Jake       Engineering
Lisa       Engineering
Sue                HR

--- (df2a) ---
          hire_date
employee
Lisa           2004
Bob            2008
Jake           2012
Sue            2014
Merge using left_index=True and right_index=True
                group  hire_date
employee
Bob         Accounting       2008
Jake       Engineering       2012
Lisa       Engineering       2004
Sue                HR       2014

'''



'''
2nd Question: Using Pandas Perform Concat and Append.
'''
import pandas as pd
data_x = {
    'A': ['A0', 'A1'],
    'B': ['B0', 'B1']
}
x = pd.DataFrame(data_x, index=[0, 1])
data_y = {
```

```
    'A': ['A2', 'A3'],
    'B': ['B2', 'B3']
}
y = pd.DataFrame(data_y, index=[2, 3])

# Assign x's index to y's index (Duplicate Indices)
y.index = x.index

print("--- Original DataFrames ---")
print("x:")
print(x)
print("\ny (now has duplicate indices [0, 1]):")
print(y)

print("\n" + "="*40)
print("--- Result of pd.concat([x, y]) ---")
print(pd.concat([x, y]))

print("\n" + "="*40)
print("--- Result of pd.concat([x, y], ignore_index=True) ---")
print(pd.concat([x, y], ignore_index=True))

'''
Output:
--- Original DataFrames ---
x:
    A   B
0  A0  B0
1  A1  B1

y (now has duplicate indices [0, 1]):
    A   B
0  A2  B2
1  A3  B3

========================================
--- Result of pd.concat([x, y]) ---
    A   B
0  A0  B0
1  A1  B1
0  A2  B2
1  A3  B3

========================================
--- Result of pd.concat([x, y], ignore_index=True) ---
    A   B
0  A0  B0
1  A1  B1
2  A2  B2
3  A3  B3
'''

'''
Question 3: Demonstrate NumPy Structured arrays
'''
```

```python
import numpy as np

# --- 1. Creating a Structured Array ---
print("--- 1. Creating a Structured Array ---")
data_type = [
    ('ID', 'i4'),
    ('Product', 'U10'),
    ('Price_USD', 'f8')
]

inventory = np.array([
    (101, 'Laptop', 1200.50),
    (102, 'Monitor', 350.00),
    (103, 'Keyboard', 75.99)
], dtype=data_type)

print("Structured Array (Inventory Records):\n", inventory)
print("\nDType Structure:\n", inventory.dtype)
print("-" * 20)


print("\n--- 2. Accessing Structured Array Data ---")
print("Accessing the 'Product' field (all product names):")
products = inventory['Product']
print(products)
print(f"Result Type: {type(products)}, DType: {products.dtype}")
print("-" * 50)

print("Accessing the record at index 1 (Monitor):")
record_1 = inventory[1]
print(record_1)
print(f"Result Type: {type(record_1)}, DType: {record_1.dtype}")
print("-" * 50)

print("Accessing 'Price_USD' from the record at index 0 (Laptop's Price):")
price = inventory[0]['Price_USD']
print(price)

'''
Output:
--- 1. Creating a Structured Array ---
Structured Array (Inventory Records):
 [(101, 'Laptop', 1200.5 ) (102, 'Monitor',  350.  )
 (103, 'Keyboard',   75.99)]

DType Structure:
 [('ID', '<i4'), ('Product', '<U10'), ('Price_USD', '<f8')]
--------------------

--- 2. Accessing Structured Array Data ---
Accessing the 'Product' field (all product names):
['Laptop' 'Monitor' 'Keyboard']
Result Type: <class 'numpy.ndarray'>, DType: <U10
--------------------------------------------------
Accessing the record at index 1 (Monitor):
```

```
(102, 'Monitor', 350.0)
Result Type: <class 'numpy.void'>, DType: [('ID', '<i4'), ('Product', '<U10'), ('Price_USD', '<f8')]
--------------------------------------------------
Accessing 'Price_USD' from the record at index 0 (Laptop's Price):
1200.5
'''



'''
Question 4: Demonstrate 3 different examples of Broadcasting
'''
import numpy as np

print("--- Example 1: ---")
M = np.ones((2, 3))
a = np.arange(3)
print("M + a: ")
print(M + a)


print("--- Example 2: ---")
a = np.arange(3).reshape((3, 1))
b = np.arange(3)
print("a + b: ")
print(a + b)

print("--- Example 3: ---")
M = np.ones((3, 2))
a = np.arange(3)
print("M + a[:, np.newaxis] =")
print(M + a[:, np.newaxis])

'''
Output:
--- Example 1: ---
M + a:
[[1. 2. 3.]
 [1. 2. 3.]]
--- Example 2: ---
a + b:
[[0 1 2]
 [1 2 3]
 [2 3 4]]
--- Example 3: ---
M + a[:, np.newaxis] =
[[1. 1.]
 [2. 2.]
 [3. 3.]]
'''


'''
Question 5: On a given dataset perform Aggregations
'''
import numpy as np
```

```

data = np.array([5, 12, 18, 12, 5, 20, 25, 18, 12])

aggregations = {
    "1. Minimum (Min)": np.min(data),
    "2. Maximum (Max)": np.max(data),
    "3. Mean (Average)": np.mean(data),
    "4. Median (Middle Value)": np.median(data),
    "5. Standard Deviation (Std)": np.std(data),
    "6. Variance (Var)": np.var(data)
}

print(f"Dataset Array: {data}")
print("-" * 40)
print("Statistical Aggregations (NumPy Only):")
print("-" * 40)

for name, value in aggregations.items():
    print(f"{name}: {value:.2f}")

'''
Output:
Dataset Array: [ 5 12 18 12  5 20 25 18 12]
----------------------------------------
Statistical Aggregations (NumPy Only):
----------------------------------------
1. Minimum (Min): 5.00
2. Maximum (Max): 25.00
3. Mean (Average): 14.11
4. Median (Middle Value): 12.00
5. Standard Deviation (Std): 6.35
6. Variance (Var): 40.32
'''

'''
Question 6: Demonstrate Universal Functions
'''
import numpy as np

# Set print options for clean, concise output
np.set_printoptions(precision=4, suppress=True)

print("="*20)
print("NumPy Universal Functions (Ufuncs) Demonstration")
print("="*20)

X = np.arange(5)
Y = np.arange(5, 10)
A = np.array([-2.5, 0.0, 1.0, np.pi / 2, -1.0])

print(f"Input Array X: {X}")
print(f"Input Array Y: {Y}")
print(f"Input Array A: {A}\n")

print("--- 1. Basic Arithmetic (Binary) ---")
```

```
print(f"X + Y (np.add): {X + Y}")
print(f"Y - X (np.subtract): {Y - X}")
print(f"X * Y (np.multiply): {X * Y}")
print(f"Y / (X + 1) (np.divide): {np.divide(Y, X + 1)}")
print(f"Y % 3 (np.remainder): {Y % 3}\n")

print("--- 2. Absolute Value (Unary) ---")
absolute_A = np.abs(A)
print(f"np.abs(A):\n{absolute_A}\n")

print("--- 3. Trigonometric Functions (Unary) ---")
sine_A = np.sin(A)
print(f"np.sin(A): {sine_A}")

cosine_A = np.cos(A)
print(f"np.cos(A): {cosine_A}")

tangent_A = np.tan(A)
print(f"np.tan(A): {tangent_A}")

A_clipped = np.clip(A, -1, 1) # Ensure values are within [-1, 1] range
arcsin_A = np.arcsin(A_clipped)
print(f"np.arcsin(A_clipped): {arcsin_A}\n")

print("--- 4. Exponents and Logarithms (Unary) ---")
exp_X = np.exp(X)
print(f"np.exp(X) (e^x):\n{exp_X}")

log2_Y = np.log2(Y)
print(f"np.log2(Y):\n{log2_Y}")

log_Y = np.log(Y)
print(f"np.log(Y):\n{log_Y}")

print(f"np.expm1(X * 0.0001):\n{np.expm1(X * 0.0001)}")

'''
Output:
==================
NumPy Universal Functions (Ufuncs) Demonstration
==================
Input Array X: [0 1 2 3 4]
Input Array Y: [5 6 7 8 9]
Input Array A: [-2.5     0.      1.      1.5708 -1.    ]

--- 1. Basic Arithmetic (Binary) ---
X + Y (np.add): [ 5  7  9 11 13]
Y - X (np.subtract): [5 5 5 5 5]
X * Y (np.multiply): [ 0  6 14 24 36]
Y / (X + 1) (np.divide): [5.     3.     2.3333 2.     1.8   ]
Y % 3 (np.remainder): [2 0 1 2 0]

--- 2. Absolute Value (Unary) ---
np.abs(A):
[2.5    0.     1.     1.5708 1.    ]
```

```
382
383 --- 3. Trigonometric Functions (Unary) ---
384 np.sin(A): [-0.5985  0.      0.8415  1.     -0.8415]
385 np.cos(A): [-0.8011  1.      0.5403  0.      0.5403]
386 np.tan(A): [ 7.4702e-01  0.0000e+00  1.5574e+00  1.6331e+16 -1.5574e+00]
387 np.arcsin(A_clipped): [-1.5708  0.      1.5708  1.5708 -1.5708]
388
389 --- 4. Exponents and Logarithms (Unary) ---
390 np.exp(X) (e^x):
391 [ 1.      2.7183  7.3891 20.0855 54.5982]
392 np.log2(Y):
393 [2.3219 2.585  2.8074 3.     3.1699]
394 np.log(Y):
395 [1.6094 1.7918 1.9459 2.0794 2.1972]
396 np.expm1(X * 0.0001):
397 [0.     0.0001 0.0002 0.0003 0.0004]
398 '''
399
400 '''
401 Question 7: Demonstrate Operating on Null Values
402 '''
403 import pandas as pd
404 import numpy as np
405
406 print("="*30)
407 print("Scenario 1: Dropping Columns that are COMPLETELY Null")
408 print("="*30)
409
410 data_drop = {
411     'ID': [101, 102, 103],
412     'Value': [45, 67, 88],
413     'Notes': [np.nan, np.nan, np.nan],
414     'Comments': ['Good', None, 'Great']
415 }
416 df_drop = pd.DataFrame(data_drop)
417
418 print("--- Initial DataFrame (df_drop) ---")
419 print(df_drop)
420
421 null_col_check = df_drop.isnull().all()
422 print("\n--- Check: df_drop.isnull().all() ---")
423 print(null_col_check)
424
425 df_dropped = df_drop.dropna(axis=1, how='all')
426
427 print("\n--- Result: DataFrame after dropping ALL-null columns ---")
428 print(df_dropped)
429
430 print("="*30)
431 print("Scenario 2: Filling Partial Null Values with the Scalar 0")
432 print("="*30)
433
434 df_fill = df_dropped.copy()
435
436 print("--- DataFrame before filling (df_fill) ---")
```

```
print(df_fill)

print("\n--- Null value count per column before filling ---")
print(df_fill.isnull().sum())

scalar_value = 0
df_filled = df_fill.fillna(value=scalar_value)

print(f"\n--- Result: DataFrame after filling nulls with '{scalar_value}' ---
")
print(df_filled)


'''
Output:
==================================
Scenario 1: Dropping Columns that are COMPLETELY Null
==================================
--- Initial DataFrame (df_drop) ---
    ID  Value  Notes Comments
0  101     45    NaN     Good
1  102     67    NaN     None
2  103     88    NaN    Great

--- Check: df_drop.isnull().all() ---
ID          False
Value       False
Notes        True
Comments    False
dtype: bool

--- Result: DataFrame after dropping ALL-null columns ---
    ID  Value Comments
0  101     45     Good
1  102     67     None
2  103     88    Great
==================================
Scenario 2: Filling Partial Null Values with the Scalar 0
==================================
--- DataFrame before filling (df_fill) ---
    ID  Value Comments
0  101     45     Good
1  102     67     None
2  103     88    Great

--- Null value count per column before filling ---
ID          0
Value       0
Comments    1
dtype: int64

--- Result: DataFrame after filling nulls with '0' ---
    ID  Value Comments
0  101     45     Good
1  102     67        0
```

```
2  103     88    Great


'''
```